2002

Fast scalable visualization techniques for interactive billion-particle walkthrough

Xinlian Liu
Louisiana State University and Agricultural and Mechanical College

FAST SCALABLE VISUALIZATION TECHNIQUES FOR INTERACTIVE BILLION-PARTICLE WALKTHROUGH

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in

The Department of Computer Science

by
Xinlian Liu
B.E., Huazhong University of Science & Technology, 1993
August 2002

ACKNOWLEDGEMENTS

This dissertation would not be possible without my major professor Dr. Aiichiro Nakano. It is difficult to overstate my gratitude for his guidance, patience, enthusiasm, encouragement, and persistent support when I needed them most. It is equally an honor and a blessing for being his student. For me, Dr. Nakano is not only a career role model, but also a light tower in my life. Should I have an academic career myself, I would hope to share with my students the same care, love and passion I received from him.

I would also like to thank Dr. Priya Vashishta and Dr. Rajiv Kalia for providing me the research and computing resources of Concurrent Computing Laboratory for Materials Simulations (CCLMS). I could not thank enough Dr. Jerry Trahan for his penetrating questions on my proposal defense and unconditional assistance during final defense. A lot of thanks go to Dr. S. S. Iyengar and Dr. John Tyler, not only for serving in my committee, but also for invaluable advice throughout my stay in the Department of Computer Science at LSU. Many thanks should be given to members of the visualization group at CCLMS, especially Ashish Sharma, Paul Miller, and Wei Zhao for their comradeship, criticism and inspirations. Dr. Sanjay Kodiyalam and Sharma Ashish spent unstinting time helping me going through my thesis and presentations.

Special thanks to Dr. Han Chen and couple of David Huelsbeck and Cate Mallory for helping me choose LSU. Some very nice southern people have made my living in Baton Rouge a true pleasure. They include Ms. Virginia Grenier at the International Hospitality Foundation, our host family Joes. Ms. Susanna Dixon, editor at the Graduate School went out of her way to help me on the graduation arrangement. Dr. Bush Jones and Dr. Don Kraft

are always there when I need their attention. Dr. Sukhamay Kundu's lectures gave me a lot of enjoyment; the ride on his old Datsun is equally amusing. Dr. David Pollock gave me a taste of bioinformatics in his protein evolution class. I would also extend my gratitude to all teachers in my college and grade schools years. Thank you.

This dissertation is dedicated to my Dad, who has been fighting and winning the battle against cancer in the past two years. Thank you Mom, for leading this battle; and thank you sister, for filling my filial role when I can't be there. Thank you Grandma, I always know you are the person who loved me most in this world, and that has been the strength that holds me up in time of adversity. The strong family tie that we share is the greatest treasure. Lastly, thank you Zhaojing, my wife, for all your love and understanding in the last five years. Wish you good luck for your defense next year.

Life is beautiful.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

This research develops a comprehensive framework for interactive walkthrough involving one billion particles in an immersive virtual environment to enable interrogative visualization of large atomistic simulation data. As a mixture of scientific and engineering approaches, the framework is based on four key techniques: adaptive data compression based on space-filling curves, octree-based visibility and occlusion culling, predictive caching based on machine learning, and scalable data reduction based on parallel and distributed processing. In terms of parallel rendering, this system combines functional parallelism, data parallelism, and temporal parallelism to improve interactivity.

The visualization framework will be applicable not only to material simulation, but also to computational biology, applied mathematics, mechanical engineering, and nanotechnology, *etc.*

CHAPTER 1

INTRODUCTION

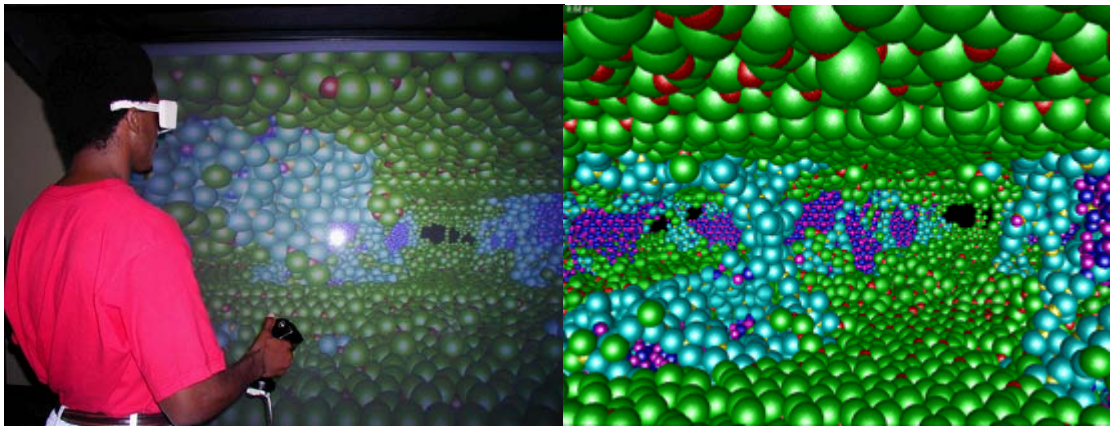
1.1 Statement of the Problem

There are three branches in scientific research: i) analytical theory; ii) experiment; and iii) computer simulation [1]. In the field of simulation, visualization plays an important role to provide a deeper understanding of information and data [2]. In particular when the dataset is large and complex, it usually makes no sense for scientists to go through data consisting of billions of figures. However, visualization of large datasets remains a challenge [3]. This doctoral research will focus on a problem in which an observer is immersed in an environment and maneuvering around to visually discover patterns.

We define the real-time walkthrough problem as follows: given a three dimensional space W, and a virtual camera controlled by a human moving inside W on a path p at speed v, generates motion pictures for the camera to transmit to proper display equipment. The human controller, based on what he or she sees from the display equipment, specifies path p interactively.

A typical walkthrough in materials simulation is shown in Fig. 1.1 [4]. At left, a researcher is immersed in an atomistic model of a fractured ceramic-fiber nanocomposite in ImmersaDesk; right, a detailed scene shows fibers that are pulled out during fracture. ImmersaDesk is a portable multi-person, high-resolution, 3D video and audio environment [5]. The researcher is using a 3D mouse called "wand" (in his left hand) to move around, like steering a car, in the 3D environment. The 3D glasses he wears synchronize with the

display to give him a stereo effect. A sensor attached to the 3D glasses tracks the position and orientation of his head, so that the scene can be updated accordingly. The computer that drives the ImmersaDesk is an SGI Onyx2 graphics server. Onyx2 is capable of both supercomputing and visualization and is designed to simultaneously process 3D graphics, imaging, and video data in real time. Onyx2 has peak performance of outputting 13.1 million polygons[1] in one second.



(a) **(b)**

Figure 1.1: (a) A researcher immersed in an atomistic model of a fractured ceramic nanocomposite (silicon nitride matrix reinforced with silica-coated silicon carbide fibers) in an immersive and interactive ImmersaDesk, which is connected to and driven by an SGI Onyx2 graphics workstation. Small spheres represent silicon atoms, and large spheres represent nitrogen (green), carbon (magenta), and oxygen (cyan) atoms. (b) A detail scene shows the fibers are pulled out during fracture.

Shown in Fig. 1.1 is a fiber-reinforced silicon nitride system that contains 1 million atoms (only part of the entire system is visible in this frame). However, to render 1 million atoms, each represented as a sphere approximated with 50 polygons, about 50 million polygons need to be rendered for one scene. To maintain a moderate refreshing rate of 30 frames per second, we need to render 1.5 billion polygons per second, which far exceeds the capability of most advanced graphics cards (see discussion in §1.2) available today.

[1] Polygons are the basic unit graphics cards use to represent objects. The number of polygons involved often determines how close this approximation reflects objects being represented.

This technological gap between atomistic simulations and visualization is increasing rapidly. For example, state of the art simulations at the Concurrent Computing Laboratory for Materials Simulation (CCLMS) at Louisiana State University involve one billion atoms [6].

To overcome this gap, this dissertation research will develop a framework for real-time walkthrough of a billion atoms by combining novel data-structure, algorithmic and hardware solutions.

The conceptual framework of the project can be used in areas where an extremely large volume of structured/unstructured data needs to be visualized. These areas include computational science, chemistry, biology, as well as applied mathematics, statistics, education, and economics. For example, improved weather forecast accuracy will have societal payoffs if meteorologists can visualize the massive amount of data obtained from observation and simulation [7]. Despite enormous scientific advances in the past few decades, the difficulty of handling and understanding massive datasets still hampers scientific progress. The limiting factor is human understanding of massive amounts of data. Results of this project will also be essential in developing an immersive user interface to remote massively parallel simulations.

The following estimation illustrates the challenge of this project. Suppose there are a billion atoms in the user's field of view. To render an atom as a sphere, at least 20 polygons are required, which can be seen in Fig. 1.2 that shows the visual effect of a sphere approximated by different numbers of polygons. For a billion atoms, this amounts to 2×10^{10} polygons.

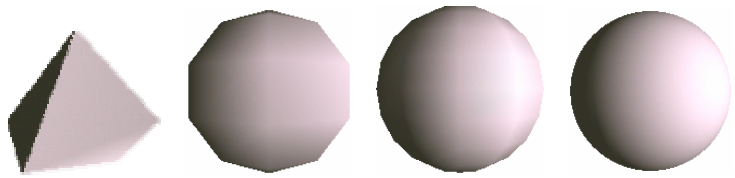


Figure 1.2: A sphere approximated with various numbers of polygons: 6, 25, 100 and 2500.

To have a smooth transition between frames so that the objects on the screen will not jump, we need at least 20 frames[2] per second [8]. Then the number of polygons that need to be rendered per second is $20 \times 10^{10} \times 20 = 4 \times 10^{11}$. (Although not all the polygons will be visible, as the great majority of them are either blocked by other atoms or are located at the invisible side of visible atoms, the graphic card still needs to render them.) A state of the art graphics card can deliver only 100 million[3] polygons per second, and therefore, brute-force rendering would result in a discrepancy between the requirement and hardware capability of a factor 4000. In fact, to output the polygons to the screen is the simplest task that a graphic card performs. The most time consuming task is to calculate the light and shade for all the possible reflections and refractions for all the polygons. Due to this challenge, it is impossible to output the image to a display device within the requested time constraint, even if we have everything prepared ready in the memory. To make the situation worse, we cannot have everything ready in the video memory on the graphic card because of the limited size of memory.

We have conducted an experiment on an SGI Onyx2, which is capable of rendering 13.1 million polygons per second with 64MB texture memory. When we tried to render a 1 million-atom configuration through the pipeline, the computer froze because the computation

[2] NTSC standard TV and theater movie have frame rate of 30 and 24 frames per second, respectively.
[3] Expected data of GeForce4 GTS Ultra (nVidia code name NV20) expected to be released May 2001. Dreamcast console can render 18 million polygons per second; Sony Playstation 2 can render 75 millions polygons per second; Sony Playstation 3 (to be released 2002) 750 millions polygons per second, 256MB memory, internal bus 48Gbps.

involved kept it busy and thereby stopped responding to any input from the keyboard or mouse.

These estimations show that currently there is no hardware that can support the goal stated above. What is needed is a strategy combining software, hardware as well as computer and network architecture solutions.

1.2 Background and Literature Review

Raw data generated by scientific simulation goes through a graphics pipeline before it is output to a display equipment, which can be either a traditional monitor or a specialized device such as the immersive and interactive 3D projection systems CAVE or ImmersaDesk [5]. This graphics pipeline is illustrated in Fig. 1.3.

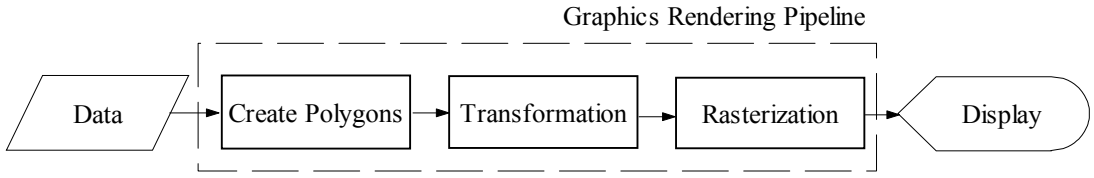


Figure 1.3 Graphics rendering pipeline. Data generated by scientific computing are rendered in three steps: i) polygons are generated; ii) they are translated to correct position; and, iii) they are transferred into 2D pixels, the form that we see on the screen. Together they form the graphics-rendering pipeline.

In Fig. 1.3 the graphics pipeline is enclosed by the dashed rectangle. The graphics pipeline usually refers to a graphics accelerator card installed on a computer, and its speed is limited by the speed of the image processor[4] on the card. As in the case of the heart of the computer, the Central Processing Unit (CPU), the processing capability of this chip is limited by integrated circuit technology, and there is a limit on how fast a single pipeline can perform.

[4] This processing chip is sometimes referred as GPU (Graphics Processing Unit).

One way to overcome this bottleneck is parallel graphics rendering. One common way of achieving parallization is dataset decomposition, the method of decomposing the dataset into smaller chunks, so that they can be assigned to multiple graphics processors. The data is then processed in parallel to achieve speedup.

Funkhouser's group at Princeton University has a project called 'High Performance Parallel Rendering on a PC Cluster' (Samanta 2000), which is a typical implementation of the parallization schema explained above by employing a three-layer architecture. The dataset is decomposed into subsets in the first layer, and multiple graphics servers render the subsets separately in the second layer. Finally, the third layer assembles the rendered sub-scenes into the full scene. Their focus is on the very end stage of the graphics-rendering pipeline without addressing the visual interactions between two objects that belonging to different subsets [9]. In other words, their interest is the feasibility of the parallel organization rather than the graphics rendering. However, the latter is one of the key problems we are facing.

Hanrahan's group at Stanford University has employed a two-layer approach [10]. Instead of assembling sub-scenes into a full scene by a graphics computer, they output tiled sub-scenes directly to a large tiled display, *i.e.*, a monitor wall or a matrix of projectors. They have also constructed a high-efficiency low-cost system by using commodity PC graphics cards such as the nVidia GeForce2 GTS[5]. The graphics performance of these systems is comparable to that of high-end graphics workstations. To facilitate the co-operation of graphics servers for parallel rendering, they have also attempted to reduce network traffic in remote visualization by using an efficient packing scheme for OpenGL codes and integrated it with network transportation [11].

[5] One of most popular graphics cards among the PC computer gaming industry.

Another parallel scheme uses multiple computers to greatly reduce the data that are fed to the pipeline instead of using multiple computers for parallel rendering. By reducing the data that is sent to the pipeline, even a single pipeline is expected to be able to handle a massive workload in a reasonable time.

The Visualization and Graphics Research Group lead by Hamann at the University of California at Davis has a project of visualization over a wide area network [12, 13]. A data structure of a time-space partition (tsp) tree has been proposed to capture the coherence in both spatial and temporal changes. They proposed a complete pipeline schema with compression techniques to reduce the cost of network transfer. Their system has been tested on a PC cluster in Japan as well as on an SGI Origin 2000 at NASA Ames Research Center with the display located at UC Davis.

To alleviate the burden of the last rendering stage, Crawfits *et al.* at Ohio University have introduced the idea of pre-processing. They suggested for 'Image Based Rendering Assisted Volume Rendering (IBRAVR)' [14] that large data is partially pre-rendered on a large computational engine close to the data, then final rendering is performed on a workstation. The pre-rending reduces the load on the rendering stage, especially if the rendering is performed in a rather low-performance graphics computer such as a personal computer.

The Visapult project [14-16] at the University of California at Berkeley has exploited the IBRAVR idea for developing a prototype application and framework for remote visualization for large-scale scientific datasets. Visapult consists of two components: a viewer and a back-end pre-renderer. The back-end is a parallel application that loads large

scientific datasets using domain decomposition and performs software rendering[6] on each sub-domain, producing layered images of sub-domains. The viewer, also a parallel application, implements IBRAVR to re-combine images produced by the back-end. Their current project involves terabyte visualization, the largest as far as we know.

The decomposing methods discussed above have been developed mainly for volume rendering, which is the major representation method in fields such as medical image processing, where body tissues are represented by the accumulation of semi-transparent image layers[7]. However, the problem addressed by these projects is different from ours: For volume rendering, the datasets can be sliced in any way and the parallization is simple because i) occlusion culling is not involved, and ii) different parts of the scene do not intervene.

Other walkthrough projects are concerned with 3D modeling, in which the spatial space is comprised of objects with shape, such as furniture, buildings and terrain. For this type of problem, only the outside surfaces need to be rendered; however, cross-boundary objects need special care upon rendering.

The Walkthru Project [17] at the University of North Carolina at Chapel Hill focuses on 3D modeling. They use the Hierarchical Levels of Detail (HLODs) technique to reduce the dataset by showing only parts that are in the current field of view [18]. The largest dataset they have tested is a Tanker comprised of 82 million triangles. They have developed fast rendering methods specifically tailored for triangle-based 3D objects [19].

[6] Software rendering refers to the method that, instead of rendering a scene by letting data goes through the graphics pipeline, generating the scene by tracing the path of light stream.
[7] These kinds of data are created by layer-by-layer diagnostic methods such as Computerized axial Tomography (CT).

The above approaches have been developed for volumetric and 3D modeling data and are not applicable to atomistic data in our project. Also, some of them emphasize on the front-end of the graphics rendering pipeline and others the back-end, and there is no comprehensive solution for extremely large volume atomistic datasets scientific visualization. This goal can only be achieved through a combination of visualization, graphics, data storage and modeling, and networking technology, which is presented in later chapters of this thesis.

In this dissertation research, we develop a framework based on four key techniques: adaptive data compression, scalable occlusion culling, predictive caching, and hierarchical visibility culling.

1.3 Outline of the Dissertation

The outline of this dissertation is as follows. In Chapter 2, an overview of the comprehensive framework is provided. Following that, Chapter 3 describes the data compression schema; Chapter 4 talks about octree-based visibility culling; Chapter 5 explains the occlusion culling based on parallel computing; and, Chapter 6 deals with neural network based prediction schema. Results are given in Chapter 7, and Chapter 8 contains conclusions.

CHAPTER 2

REAL-TIME BILLION-ATOM WALKTHROUGH IN VIRTUAL ENVIRONMENTS

2.1 Statement of the Problem

As explained in the introduction, our goal is to render large datasets consisting of a billion atoms interactively in an immersive simulation environment, which is comprised of a massively parallel computer such as the 1,280-node IBM SP3 system located remotely at the Naval Oceanographic Office Major Shared Resource Center where simulations are performed, a 166-node PC cluster to preprocess visualization data, an SGI Onyx2 graphics server for rendering, network connections and an immersive and interactive 3D visualization environment named ImmersaDesk for projection as illustrated in Fig. 2.1.

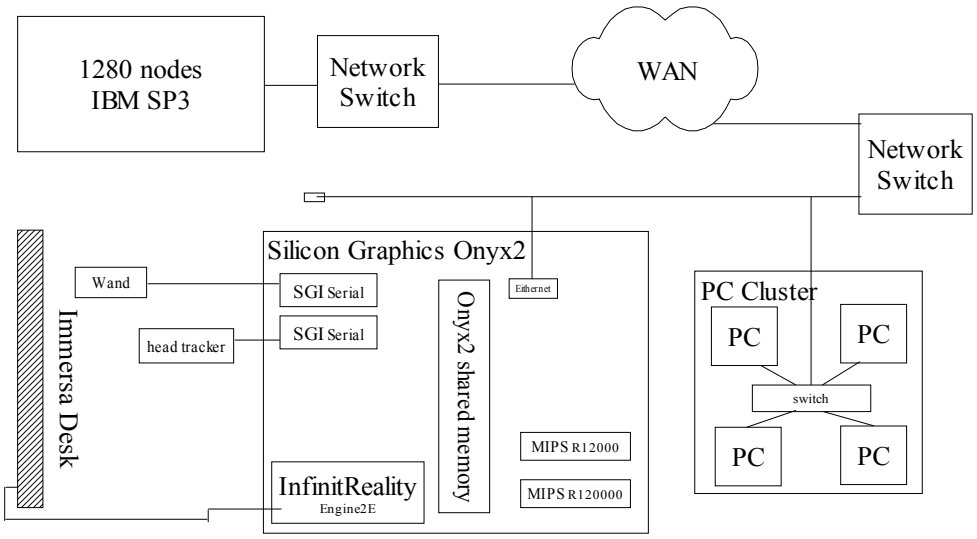


Figure 2.1: Supercomputing/Visualization environment.

ImmersaDesk is a projection system for multi-user high-resolution immersive virtual reality display. The head tracker tracks the position and orientation of the user's head and

subsequently updates the display to give the user the feeling of changing scenes according to head movements. The user, while looking at the ImmersaDesk through a pair of 3D glasses, uses the wand to direct where he/she want to go, as if walking through a space formed by the immersive projection.

The primary dataset in our project is a collection of N atoms placed in a special box of size ($L_x \times L_y \times L_z$). This is a set of all-atom data: $\{\vec{r}_i, s, \sigma^{\alpha\beta} \mid I = 1,2,\ldots N;\ s \in (Si, O, C, N,$ *etc*.$);\ \vec{r}_i = (x_i, y_i, z_i);\ 0 \le x_i \le L_x;\ 0 \le y_i \le L_y;\ 0 \le z_i \le L_z;\ \alpha\beta \in (xx, yy, zz, xy, yz, xz)\}$. For each atom i, the following information are stored: (x_i, y_i, z_i) are the three-dimensional coordination of atoms; s represents atomic species, and $\sigma^{\alpha\beta}$ are stress components.

The goal of this project is to achieve interactive response speed (≤1 second per frame) to render an updated scene according to the user's movement in the immersive environment. The objects to be rendered in this project are balls[8]. Because one of the goals of the simulation is to identify atomistic processes for which we do not have *a priori* structural information, it is not straightforward to simplify off-center parts or details.

The extremely large volume of data involved slows down traditional visualization methods, and consequently makes them impractical. The limitation is on every aspect in the computing environment: physical memory, cache size, bus bandwidth and network throughput, *etc*. Therefore, the problem cannot be solved by tune-ups, but instead re-invention of the engine itself is required.

[8] To illustrate the complexity of the problem of effectively rendering a solid ball, here is the list of the factors that need to be addressed: 1) Software/hardware; 2) OpenGL or not; 3) Computer platform; 4) Bus type; 5) Rendering precision; 6) Speed requirement; and a lot more.

2.2 Overview of the Solution

Concerns in the designing phase of parallelization of rendering include the choice between hardware or software solution; and the parallel architecture, communication mechanism, and memory constraints. We propose to develop a comprehensive framework to the solution of this "billion-atom walkthrough" problem.

Figure 2.2 shows a diagram of the proposed solution: First, raw data generated by simulations on a remote supercomputer is compressed, so that they can be transferred via wide area network (WAN). On the local site, the retrieved data goes through a PC cluster, where distributed visibility culling is conducted. Subsequently, the data is transferred to the graphics-rendering computer via a neural-network-based predictive caching component.

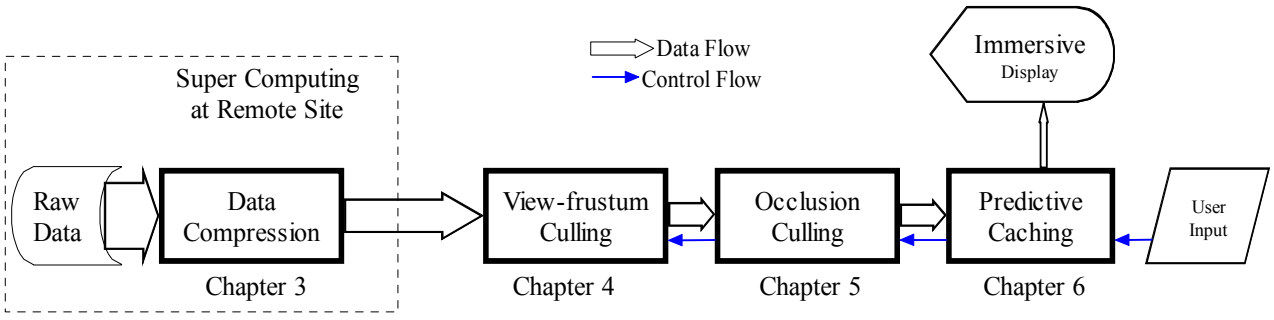


Figure 2.2: Overview of the billion-particle real-time walk-through solution. Huge amount of raw data generated by supercomputing on remote site is firstly compressed and then transferred back. A PC cluster filters out components that are invisible efficiently in parallel processing. An actively predictive caching mechanism selectively fetches data that are most likely needed next. The final rendering is done on a high performance professional graphics system.

Chapters 4 and 5 deal with datasets culling. There are three types of culling [20] involved in computer graphics rendering, as shown in Fig. 2.3. We conduct the view-frustum culling by exploiting octree data structure (Chapter 4), the occlusion culling is conducted by

hierarchical computing (Chapter 5), and the back-face culling is conducted by Z-buffer comparisons[9].

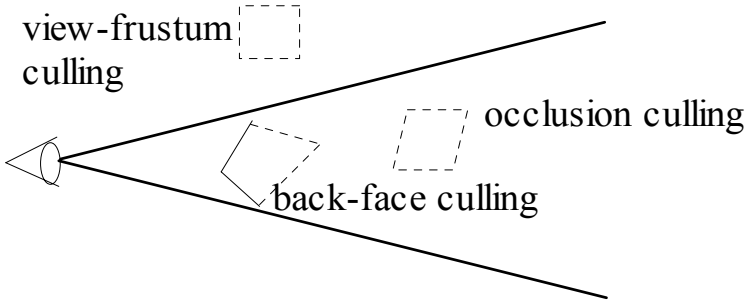


Figure 2.3: Three types of visibility culling techniques. Solid lines represent visible surfaces while dashed lines represent invisible surfaces.

2.3 Data Compression

A billion-particle simulation produces 100GB of data including species, coordinates, velocities, and stress components [21] per frame. Simulations are done at national supercomputer centers and need to be transferred, creating a great challenge for the network capacity. Data has to be reduced within reasonable time before it can be transferred via the network. This problem can be addressed using data compression. However, common compression schemes perform poorly for this specific kind of data. In the present project a compression algorithm is designed and implemented to optimize the storage of atomistic simulation data. The algorithm is based on ordering the atoms following a space-filling curve and results in significant improvement of the I/O performance.

2.4 Octree

At any given time, only a part of the whole dataset is in the effective viewing field[10] of the observer. An Octree data structure [22] is used to efficiently remove the parts that are

[9] Z-buffer comparison is done automatically by the hardware graphics engine.

out of the field of view. Each node on an octree has 8 children/sub-trees, which correspond to the sub-regions obtained by dividing a cube by cutting each surface in half in the *x*, *y*, *z* directions.

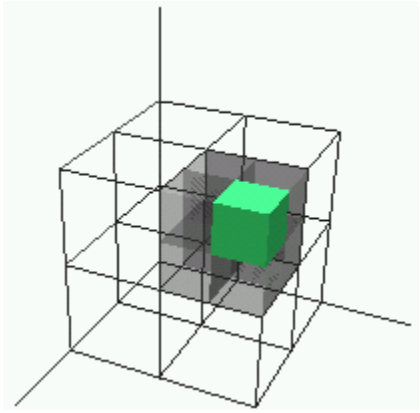


Figure 2.4: An illustration of a 3-level octree data structure. Each of the eight sub-regions is recursively divided into 8 partitions. Octree is a tree data structure with exactly 8 sub-trees or children for any non-terminal nodes. The advantage of octree data structure is that each node corresponds to a special sub-region in the application.

An Octree is an intuitive approach for clustering special objects into visible and invisible groups. If a node is invisible, then all of its sub-trees are invisible and thus can be excluded in future pipeline processing. Therefore, most invisible area can be efficiently removed from processing.

2.5 Scalable Occlusion Culling

Not all data in the given dataset will be seen at a given time during walkthrough. In fact, many atoms are either outside the viewer's field of view or hidden by other atoms. An example is that only a few upper layers of beans in a barrel are visible while the rest are hidden below. Feeding all the data into the graphics pipeline wastes more than 90%[11] of the

[10] A person's eyesight, that is how far and how close his eyes can effective focus, limits his viewing field; also the angle limits the left and right margin (usually with an angle of about 100 degrees). The effective view on the computer is also limited by the size of the screen.
[11] See the caption of Fig. 7.2.

time in rendering. We propose an algorithm that can greatly speed up the rendering process by reducing this redundancy. Because of the nature of the computation involved, this reduction itself is computation intensive. Our solution is to carry out this computation in parallel on a PC cluster. We adopt a divide and conquer strategy that divides the objects into sub-regions and allocates these sub-regions to available nodes in the PC cluster. We also adopt an efficient message exchanging method to collect the information and speed up the reduction process.

2.6 Predictive Caching (Pre-fetching)

It is intuitive that any processing on a dataset of this scale will need some caching mechanism. In some aspect, the browsing pattern is similar to locality in a memory reference pattern. That is, when a viewer (his/her mouse) goes straight for a couple of seconds, he/she will be likely to continue to go straight. However, traditional caching schemes are not suitable for the pattern of data access involved in interactive visualization. The browsing habit differs on per person, per subject, per session bases. There is not a simple set of rules on how a person will respond. A machine learning mechanism is thus expected to play an important role. We adopt a neural network based caching schema, which is capable of learning the browsing pattern from the first few movements that the person made and continuously modify it according to the user's behavior.

CHAPTER 3

DATA COMPRESSION

3.1 Background

Modern parallel computers are characterized by a technology gap resulting from the large difference in the rate of improvement of computational power and that of the I/O system [23]. CPU and RAM capabilities have increased dramatically in recent years, and the interconnecting hardware now allows combining hundreds of such processors (nodes) into powerful parallel architectures. On the other hands, the access speed has been growing at a slower rate. A data compression scheme (lossy or lossless) may be used, when possible.

Large-scale simulations usually generate huge amounts of data. Also, the simulations often run for a long period of time. For the purpose of both saving intermediate results and keeping checkpoints for failure recovery, it is favorable that more information can be saved off-line, while keeping the amount of I/O under control. Ideally, such a compression algorithm should satisfy the following requirements:

1. Significant reduction in the amount of data to be stored.
2. If the compression is lossy, the precision should be controlled.
3. Little computation and memory overhead.
4. Scalability with increasing system size.
5. Ability to handle different types of simulation data and different requirements on accuracy.

6. The algorithm should be robust and, particularly, tolerant to outliers, *i.e.*, values that do not conform to the expected range.

Data compression and data reliability are a pair of mutually exclusive factors. The simple way of cutting the precision and representing the numbers using a smaller number of bits is not feasible for the accuracy requirements of simulations. A possible approach is to use standard compression software and algorithms. However, simply running a data file through a standard compressor (Huffman or LZW) consumes a significant amount of CPU time but does not provide any benefit, since double precision data generated by many simulations contain a large number of essentially uncorrelated bits.

A space-filling curve completely fills up part of space by passing through every point in a defined order. It is intuitive to imagine that the data on the path generated by the ordering are spatially related and subsequently could be compressed by removing the redundant information.

3.2 Algorithm Analysis and Implementation

3.2.1 Octree Ordering

Several space-filling curves are known and all are defined recursively. A typical definition starts with a simple curve C_0 and prescribes how to use it to construct another, more complex curve C_1, and defines the final, space-filling curve as the limit of the sequence of curves C_0, C_1, …. [24]. Figure 3.1 shows the construction of two typical space-filling curves by successive refinement.

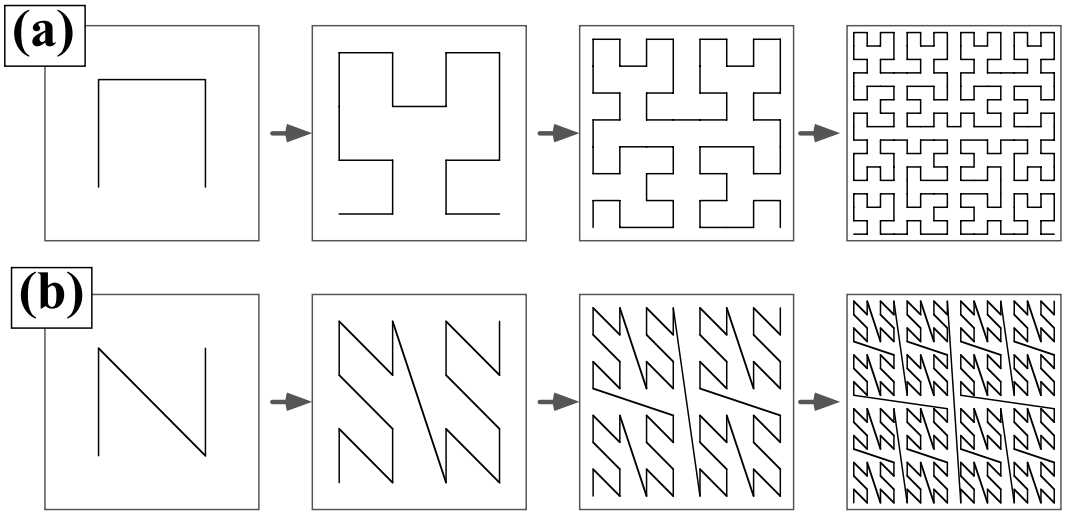


Figure 3.1: Construction of Hilbert Curves (a) and Z-Curves (b) by successive refinement.

Appropriate ordering of particles may be accomplished by arranging them according to their position on a space-filling curve. The Hilbert Curve does not contain discontinuities, but the position of an atom on this curve is not easy to compute. The Z-curve is discontinuous, but the points on this curve are easily identified using an octree index, R_i, constructed by interleaving bits of x_i, y_i, and z_i.

Let us assume that x, y, and z components of atomic positions are represented using the same number of bits, $l = \left| \log_2 \left(\max \left\{ L_x, L_y, L_z \right\} \right) \right|$, which is determined by the maximum dimension of the simulation box. The octree index R_i is then easily computed by interleaving the bits of the position components, as illustrated in the following example:

```
l = 7 bits

xi =   1   1   0   1   0   1   1
yi =  0   0   1   1   1   1   0
zi = 1   0   1   0   0   0   1
--------------------------------
Ri = 101 001 110 011 010 011 101 -- 3*7 = 21 bits
```

A potential problem arises when $l_R = l_x + l_y + l_z$ exceeds the number of bits in an integer. Assuming the size of an integer to be 32 bits (which is fairly standard on UNIX machines), it is often sufficient to represent one component of the atomic position but not necessarily all three of them. Using 64-bit integers may solve the problem but it will interfere with the portability of the code. Another possible approach is to store the octree index in a data structure that would allow arbitrary-length integer representation, but this would significantly degrade the efficiency of the code. Therefore, we have to look for other alternatives.

Let us consider plausible scenarios that would cause l_R to exceed 32 bits. The two possible reasons are:

1. The size of the system (L_x, L_y, L_z) is too large.
2. The accuracy requirements are too high, *i.e.*, the tolerance, δr, is too small.

The first case is taken care of by fragmenting the atomistic configuration into sub-domains no larger than a predefined dimension.

The second case requires further consideration. The tolerance being too small implies that the integerized atomic positions contain several low-order bits that are unlikely to be compressed. Consequently, it makes sense to split each integer into high-order bits ("head") and low-order bits ("tail"):

x_i (bit-number = l_x) --> $x_i^{(h)}$ (bit-number = $l_x^{(h)}$) and $x_i^{(t)}$ (bit-number = $l_x^{(t)}$),

so that $l_x^{(h)} + l_y^{(h)} + l_z^{(h)} \le 32$. Then, only $x_i^{(h)}, y_i^{(h)}, z_i^{(h)}$ are encoded into the octree index and output using variable-length encoding, while fixed-length encoding is applied to $x_i^{(t)}, y_i^{(t)}, z_i^{(t)}$.

Data compression can be achieved by storing only the difference (relative distance) of two adjacent particles' position rather than the absolute coordinates. Actually, it is possible to store the difference in the octree index, $\Delta R_i = R_i - R_{i-1}$, rather than ($\Delta x_i$, Δy_i, Δz_i), since R_i can be easily decoded to yield the position components.

3.2.2 Sorting

The sorting problem for the present compression scheme involves (i) sorting the octree indices, R_i, and (ii) arranging other data in the same order. To be able to handle a different data structure, the R_i are sorted separately and an "index" array is created, which is then used to rearrange other data attached to atoms. The index is defined by

```
R[i] = R0[index[i]],
```

where "*R*" and "*R0*" are the sorted and unsorted arrays, respectively. While the index array does require additional memory, the reordering of the data is performed in-place.

It is well-known that the best possible comparison-based sorting algorithm requires $\Omega(N \lg(N))$ steps. Radix-based sorting of integers scales as *O(N·l),* where *l* is the bit-number. For the octree, index *l* is likely to reach 32 (see the previous section). In atomistic simulations, it is important to maintain the *O(N)* scaling, since one may need to handle increasingly large numbers of atoms. On the other hand, computation time in radix sorting involves a large pre-factor proportional to the bit-number. Fortunately, it is possible to exploit special properties of atomistic configurations to further optimize the sorting procedure for this type of data.

As discussed in §2.1, the distribution of atoms is typically uniform, at least within sufficiently small volumes, and the atoms cannot concentrate in one place to more than a certain density. Therefore, it would make sense to tailor the sorting algorithm to be optimal for atoms scattered uniformly within the spatial range of atoms, which is equivalent to the octree indices being uniformly distributed between the appropriate minimum and maximum values.

The sorting algorithm used in the present implementation consists of two stages:

1. Sort R_i into ~N bins.
2. Sort R_i within each bin.

In the first part, N equal-size bins of size $(R_{max}-R_{min})/N$ are defined. Then, it is a simple matter to construct *O(N)* algorithm for sorting the R_i into bins. In the present implementation, the size of the bin is adjusted to the closest power of 2, so that integer divisions may be replaced by more efficient shift operations.

If the distribution of R_i is uniform, each bin is expected to contain $n \sim 1$ integers, which is the number of values to be sorted in the second stage. For efficiency, the cases of $n \leq 3$ are programmed explicitly, while $n > 3$ is handled using the heap sort algorithm. The resulting algorithm is *O(N)* for a uniform distribution and $\mathrm{O}(N \lg(N))$ in the worst-case scenario.

CHAPTER 4

OCTREE-BASED VIEW-FRUSTUM CULLING

4.1 Motivation for Visibility Culling

The OpenGL rendering engine removes hidden objects in the last stage of its graphics pipeline [25, 26], see Fig. 4.1.

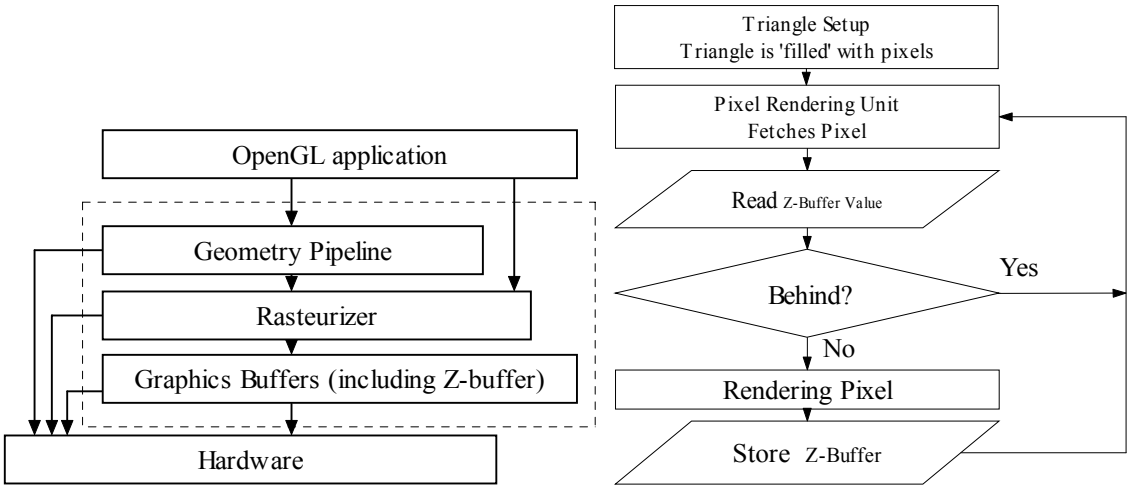


Figure 4.1: (a) OpenGL Architecture. This is a simplified illustration of the OpenGL rendering engine architecture. The compring with Z-buffer[12] is done in the last stage of pipeline rendering. Even if a pixel is not actually drawn as a result of its comparision, it has to go through all the previous stages of the pipeline and herein waste all the computational resources. (b) Actual removal procedure of a graphics card.

As shown in Fig. 4.1(a), the OpenGL hidden objects removal procedure uses Z-buffer comparison, which is detailed in Fig. 4.1(b) and is done at the very last stage of the graphics pipeline. Consequently, all the computing time for modeling, translations, coloring and lighting used for objects being removed is wasted. We can achieve a speedup in rendering by reducing the dataset before they go to the pipeline. There are two types of redundant data

[12] Z-buffer is a part of the video memory that is used to record the Z coordinates of pixels. In the OpenGL coordination system, the z axis is perpendicular to the screen. In short, OpenGL decided the visibility of an object by checking whether there are other objects between it and the viewer.

in visualization. The first is objects that are not in the effective view field and the other refers to objects that are in the view field but are hidden by peer objects between the observer and themselves. We propose to remove the first type of redundancy by utilizing an Octree data structure (see §4.2 and §4.3) and remove the second type of redundancy by scalable occlusion culling, which will be discussed in detail in Chapter 5.

4.2 Octree: Principle, Design and Implementation

Octree is a tree data structure based on a cell with eight children [27]. Each cell of an Octree represents a cube in physical space. The children can be either sub-trees or leaves. The root node of the octree represents the entire target space while each child node represents one octant of its parent.

Octree partition is a well-established technique in the computer gaming industry to reduce the number of objects that need to be updated at a given moment [28]. Recently, the Octree has been found useful in scientific applications as well. Unlike in the gaming field, where each scene is comprised of a structure of shaped objects with large empty space, the configuration we are facing is comprised of a large amount of unstructured particles uniformly located throughout the whole space. Therefore, instead of partitioning the space by objects, we partition the space by volume.

In the present problem, the data structure is designed as follows. The entire configuration is broken up recursively into 8 spatial sub-sets. The particle coordinates and associated atomic attributes are stored in an array with each member storing a few hundreds of particles, the specific number depending on the levels of octree used and the size of the dataset. The octree acts as a data structure that stores the array indices. This is shown in Fig. 4.2. The special structure of the Octree is stored linearly and continuously in the physical

memory. The address of one particle node of the Octree can be directly computed from its logical position on the Octree.

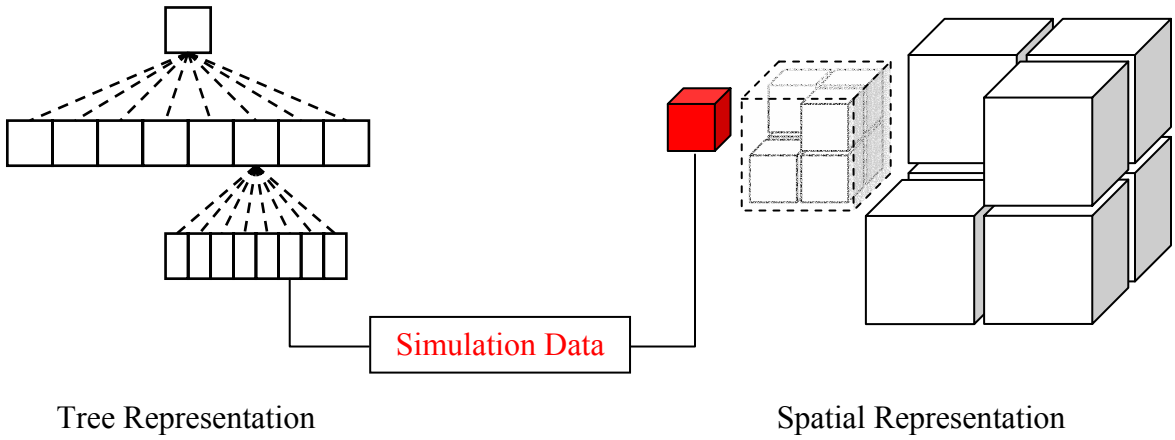


Figure 4.2: Correspondence between octree data management and the actual data storage in the memory. Structural data of the Octree is stored linearly and continuously in the memory, the address of which can be computed directly.

4.3 Octree Based Data Extraction

Data that goes to the graphics pipeline is generated in two steps: the initial extraction and subsequent refinement [29]. In the first step, the octree is traversed and an approximate region of interest is obtained based on the coordinates of the user's viewpoint. Terminal nodes of the octree are approximated as spheres, while the depth of field of the viewer constitutes a global sphere. To determine which of the node spheres are visible is a trivial mathematical computation of sphere intersection between these spheres and a global sphere with radius of viewer's depth of field originated at the user's position. The second step includes a series of visibility cullings to refine the approximation to the viewer's viewing frustum. The number of regions that are ultimately displayed depends upon three factors:

1. The number of regions that lie within the clipping shapes based on the position and orientation of the primary camera,

2. The desired maximum/minimum number of visible particles, and
3. The desired minimum frame rate.

Regions are selected using a recursive procedure, by applying clipping shapes/volumes to the system at various levels of the octree data structure. Beginning with a child of the root node, we begin this process by examining whether the center of this region is within the global sphere. If this center point lies within, we continue to apply this clipping to all of its children in the octree.

If this center point passes all the clipping tests, then we assume that some part of that region lies within our field of view. The test is then repeated for each of the 8 child nodes contained under that region if it is not a leaf node. If it is a leaf node of the octree, we consider some part of that node to be visible, and it is added to the list of nodes to be displayed during the next display cycle.

Following the selection of the regions within the clipping volumes, we sort these regions in ascending order according to their distance from the viewer. This is done to ensure that the regions displayed first are those closest to the viewer, in the case that not all the selected regions get displayed. A side product of this sorting is the implementation of multi-resolution schema, in which the atoms are rendered as spheres or spherical approximations at varying "resolutions" based upon their distance from the primary camera.

CHAPTER 5

SCALABLE OCCLUSION CULLING

5.1 Hidden Objects Removal in Computer Graphics

Occlusion culling is to remove the objects that are blocked by other objects in the viewer's view frustum from the dataset before they go through the graphics pipeline.

Various algorithms [30] for hidden object removal have been suggested since the 1960s. These include:

1. Z-Buffer (Depth-Buffer) Algorithm: This algorithm is used in the graphics pipeline as discussed in §4.1.
2. Ray Casting Algorithm: A brute force method to trace 'rays' of light originating from the view plane until they reach the eye.
3. Scan-line Algorithm: Have lines scan the view frustum and examine all polygon surfaces intersecting the scan lines to determine which is visible at each pixel along the scan line.
4. Painter's Algorithm: Sort polygon surfaces in order of decreasing depth. If there is overlap between two surfaces, split them into smaller pieces along the edge of intersection and continue the comparison.

Since we are concerned with real-time walkthrough, the algorithm to be used for hidden object removal has to be highly efficient and involve minimum computation. Any computation of distance or detection of intersection is expensive in this context, and therefore

should be avoided when possible. This restriction excludes traditional hidden objects removal algorithms mentioned above. When the user is moving around the configuration, there is no fixed projection plane, therefore, it is difficult to keep a sorted list of atoms, and it is expensive to generate one for each frame.

Considering the resolution of any display equipment is limited, once we have an *m* by *n* grid mapping the eye plane depending on display resolution. Particles are thrown to the eye plane one by one. Each particle is assigned a linear scaling factor based on its position in the *z* direction simulating perspective projection. Nearest neighbors, which are approximated by the rectangle square surrounding the sphere footprint, of any given particle on that grid can be directly computed from the *x* and *y* coordination of the atom and the scaling factor. Eight points on the outer peripheral of the rectangle square were tested. If any of the eight points turned out to be visible, then the atom will not be culled. Time complexity of occlusion is a linear function of the total number of particles.

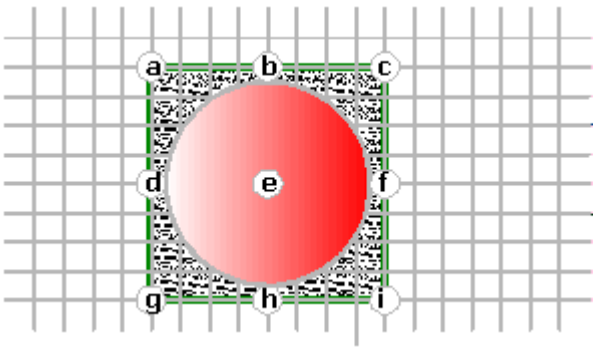


Figure 5.1: A particle is projected on the eye plane. If all points (a, b, c, d, e, f, g, h, i) are blocked by other particles, then this particle will be occluded.

A pseudo code for the occlusion culling algorithm is given below:

```
get device resolution
cover eye-plane with grid depends on resolution;
for each particle
  begin
    perform world-eye transformation;
    project the atom at eye-plane;
    calculate the footprint size;
    check if the footprint occupies empty grids;
    // only adjacent neighbors are compared
    register the atom if any part of the ball is visible;
  end;
```

5.2 Hierarchical Structure and Scalable Algorithm

The hidden objects removal algorithm described in the previous section is highly efficient. However, considering the total amount of particles involved, which is on the order of a billion, parallel computation is needed to achieve real-time performance. Since the dataset is too large to fit into any single compute node, a divide and conquer scheme is applied. That is, the job is equally divided and assigned to every node available in the parallel computer. The intermediate results from individual nodes are collected and exchanged with their neighbors in the same manner of global reduction in a hypercube computer. For an example of 4 nodes (coded two digits from '00', '01' to '10', '11'): in the first step, those nodes that have the same first digit exchange information and store the processed (winning) information in the node that has a '1' in the second digit. That node is designated as the winner of the exchange. In the second step, the two winners exchange step information and store the information in the node that has a '1' in the first digit. By doing so, the node '11' has the winning information in 2 steps. Figure 5.1 illustrates this exchanging process in a 4 node (2 dimensions) situation.

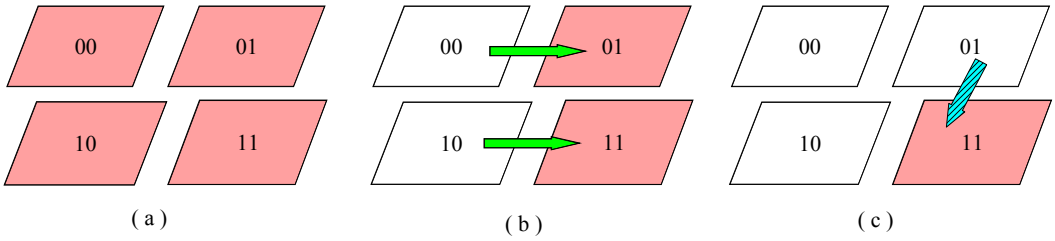


Figure 5.2: Data exchanging path in the divide-and-conquer hidden object removal algorithm. Shadowed square represents where computation takes place and the arrow illustrates data flow direction.

A similar algorithm can be obtained straightforwardly for higher dimensions. Because only one additional data exchange is involved for every doubling of size, this scheme is a scalable solution. In the extreme situation of only a single node involved, no exchange is needed. One exchange is needed for 2 nodes; for up to m nodes ($m = 2^n, n = 1,2,3,...$), at most $\log_2 m$ exchanges are needed. At later stages, most processors become idle. Although parallel efficiency is not high in this implementation, our only concern is wall-clock speedup, which is more important in real time applications. For each independent scene, up to 50,000 atoms are visible; because only the indices of atoms are transferred, network traffic capacity is not a concern. This dividing schema conforms to the octree data managing mechanism employed in the system.

Suppose the total number of particles is n and a b level octree is applied, and that particles are evenly allocated to all nodes, then the total number of nodes involved is 8^b, and each node has $n/8^b$ particles. The occlusion culling algorithm has a linear complexity of $O(n/8^b)$. Because $\log_2 8^b$ message exchanges are needed in the parallel scheme, the total time complexity is $O(\frac{n}{8^b} \times \log_2 8^b) = O(\frac{n}{8^b} \times 3b) = O(\frac{1}{c_0} \times n)$, where $c_0 = \frac{8^b}{3 \times b}$. In practice, b

is a small number. When the parallel computer for pre-processing is comprised of 8 nodes, $b = 1$.

CHAPTER 6

NEURAL NETWORK ALGORITHM IN VIEW FIELD PREDICTION

6.1 Introduction

Scientific visualization provides a deeper understanding of information and data generated by large scale simulations [2]. However, visualization of large datasets remains a challenge [3, 12, 17, 31].

We [32] implemented a parallel and distributed system that utilizes the computing power of a PC cluster to perform partial pre-rendering to reduce the dataset flowing into the graphics engine. In order to overcome the bottleneck of rather expensive graphics hardware, the computation is split into two parts. The compute-intensive preprocessing including viewer-frustum culling and occlusion culling is performed on the PC cluster, while the graphics processing intensive rendering is carried out on the graphics workstation. This system achieved a nearly interactive rendering speed of about 1 frame per second for a billion particle configuration.

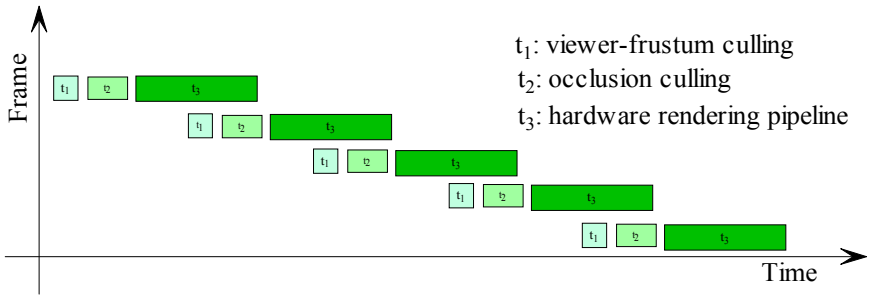


Figure 6.1: Schematic of pipeline overlapping. View frustum culling (t1) and occlusion culling (t2) are done on a PC cluster, while hardware rendering (t3) is done on an SGI graphics server.

To further improve the system performance, it is an intuitive idea to have this two-stage pipeline overlapped. For this purpose, there needs to be a mechanism to help determine which part of data should be pre-fetched to the second stage in the pipeline, which is the graphics workstation. Figure 6.1 shows this overlapping scheme.

In order to numerically solve the problem, the target space is divided into a regular 3D grid. A user's next move to 26 possible adjacent domains is coded numerically from 1 to 26. Then the problem of predicting next position is converted to that of predicting the value of a time series function. We propose an instantaneously trained neural network schema to improve the interactive speed in extremely large scale scientific visualization.

6.2 Time Series Prediction

The time series prediction problem is to forecast a future value based on knowledge of historical data. Some traditional methods, such as probabilistic method in time series function prediction, were discussed in the book by Brown [33]. One of the presumptions of these methods is that the overall distribution of the data conforms to a known probability distribution. However, as the temporal sequence considered in this research represents the trace of a human walking through a 3D space, it is difficult to define such a pattern.

There are alternative methods for time series function prediction, such as Monte Carlo methods [34] and artificial neural networks [35]. Monte Carlo methods usually involve excessive computations, which is not suitable for interactive applications. For neural networks, the obstacle lies in the training time. A prototypical neural network implementation consists of two separate phases: the (usually off-line) training phase and the on-line running phase. The need for off-line training in advance makes neural networks unsuitable for tasks that require instant learning from streaming data.

One promising method is the quick training algorithm for radial basis function (RBF) neural networks [36]. It has no local minima, and it can be trained significantly faster than backpropagation networks. However, because of the special shape of the activation function, it is not sufficiently fast for the interactive applications.

6.3 CC4 Algorithm

Kak [37] suggested a unique instant learning neural network based on a corner classification (CC) algorithm. The CC algorithm involves little computation and thus enables instantaneous training. This algorithm is suitable for fast classification problems with binary input data. Several implementations of the CC algorithm have been suggested. One of these named CC4 combines learning and generalization, and is most suitable for our problem.

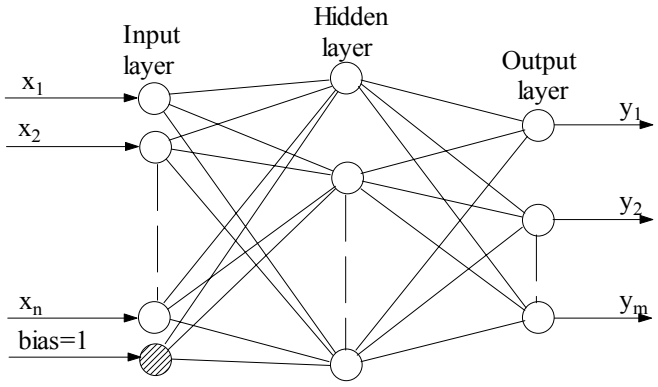


Figure 6.2: A general CC4 network architecture of three layered structure. The input layer's neurons correspond to the input vectors plus a bias neuron. The hidden layer represents the knowledge base. The output layer is the result.

A CC4 neural network consists of three layers of binary neurons: the input layer, a hidden layer, and the output layer. The number of input neurons is equal to the length of the input vector plus one bias neuron, which always has the value of 1. The number of hidden neurons is equal to the number of training samples, where each hidden neuron corresponds to

one training sample. The activation function of CC4 is as follows: The output neuron outputs 1 when the sum of all weighted inputs is greater than 0 and outputs 0 otherwise. Input layer weights are assigned as:

$$w_j = \begin{cases} 1 & x_i = 1 \\ -1 & x_i = -1 \end{cases}, w_{bias} = r - s + 1$$

For each training vector, if an input neuron receives a 1 $(x_i = 1)$, its weights to all hidden neurons are set to 1; otherwise, they are set to -1. The weights from the bias neuron to the hidden layer neurons are equal to the radius of generalization *r* minus the summation of '1's in the training vectors plus 1. Different values of *r* have been tested, and we found that the best hit ratio achieved when *r* = 1.5. Output layer weights are assigned as follows: If the training vector produces a 1 at an output neuron, the weight from its hidden neuron to that output neuron is set to 1; otherwise, it is set to -1. Figure 6.2 shows the architecture of a general CC4 network.

A pseudo code for the CC4 training algorithm is given below:

```
// w: input layer weight matrix;
// u: output layer
assign r an appropriate value,i.e. r=2;
for each training vector j do
  begin
    s = 0;
    for each input vector i do
      begin
        if training vector j gets '1'
    then s = s + 1;
        Set wi[j];
      end;
    wbias[j] := r - s + 1;
    set u[j];
  end;
```

The CC4 algorithm can be used when instant learning is desired, such as an on-line intelligent search engine [38] or short term predictions such as day trade stock values.

6.4 CC4 in the Walkthrough Problem

We have incorporated the CC4 algorithm into the interactive walkthrough system discussed in §6.1.

In order to improve the average response time, the CC4 algorithm is used to predict the next move of the user based on recent previous positions during walkthrough. Using the predicted next move, the program pre-fetches required data while the current scene is being rendered. Therefore, on the next time step, the graphics pipeline can begin rendering immediately without waiting for the data to be fetched through the network. A formal rephrase of the problem is:

$$\begin{cases} \vec{r}_{t+1} = G(\vec{r}_t, \vec{r}_{t-1}, ..., \vec{r}_{t-w+1}, H) \\ H = I(\vec{r}_{t-s+1,} \vec{r}_{t-s+2}, ..., \vec{r}_{t-1}, \vec{r}_t) \end{cases}$$

The predicted position, $\vec{r}_{t+1}$, is determined by the current $(\vec{r}_t)$ and $w-1$ previous positions, $(\vec{r}_{t-1}, ..., \vec{r}_{t-w+1})$, and knowledge of the walking pattern, *H*, which is obtained by examining the history of s previous positions. The width of testing window, *w*, and the number of training samples, *s*, are parameters to be chosen at the training stage to conform to a particular problem.

The entire space is covered with a regular 3D grid, and the smallest unit of this grid forms a cell. The user's position is discretized with the cell. Also, instead of recording absolute positions, relative positions are used, which are represented by directional numerical values as show in Fig. 6.3. All movements are represented by relative positions with cell 14 being the reference point. For example, number 1 represents a move from position 14 to position 1. A path goes from cell A to cell B (A→C→D→E→F→B) as shown in the Fig. 6.3 will be encoded as a sequence, (15,10,11,23,23), as the path shown as shadow.

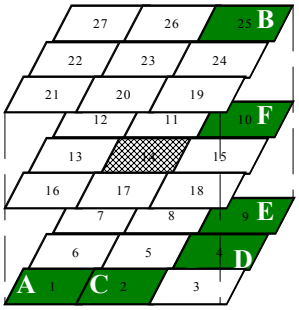


Figure 6.3: Illustration of the spatial grid. All movements are represented relative to the starting position (cell 14 is the reference cell). A path from A→C→D→E→F→B is represented as a sequence, (15,10,11,23,23).

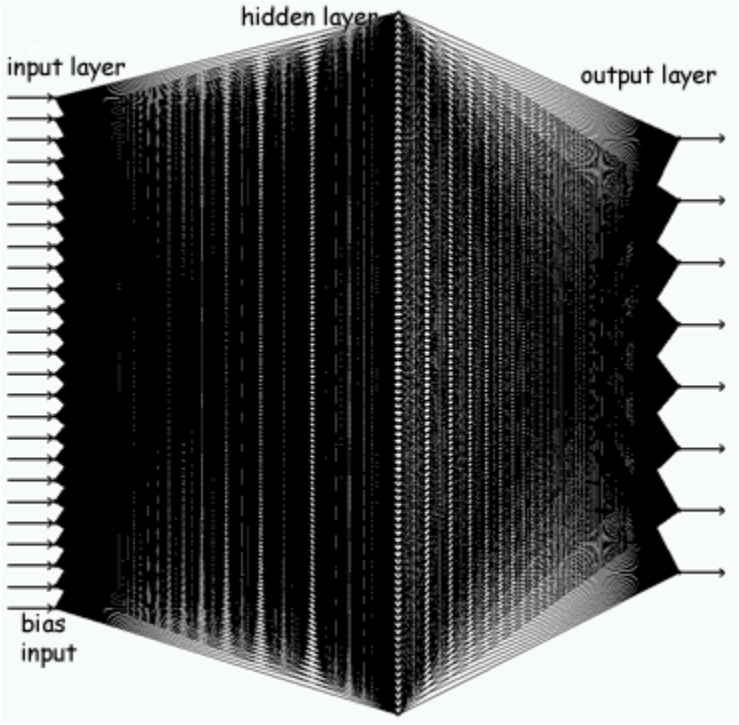


Figure 6.4: A CC4 neural network used in 2D prediction. There are 3 input vectors, each with length of 8 and 1 bias neuron amounting to 25 input neurons. The network has 100 hidden layer neurons, the number corresponding to the number of training samples. The output is the 8 directions possible directions. Neurons in adjacent layers are fully connected. For illustration purpose, 2 dimensional prediction instead of the 3 dimensional case is used because of simplicity.

Figure 6.4 illustrates a structure of a CC4 neural network for two dimensional direction predictions. The input layer has 3 input vectors, each with length of 8, representing the 8 possible directions, and a bias neuron, which brings the total number of input neuron to

25. 100 hidden layer neurons correspond to the 100 training samples, which are updated dynamically with each prediction. Output is the 8 possible directions of next move.

This predictive pre-fetching acts as 'traffic control', which is similar to the cache management mechanism. It requests future data from the previous culling procedure to feed the graphics rendering pipeline. Before generating a new frame, the rendering pipeline will run a test to check whether the required datasets reside in the memory. If the datasets exist, it will go ahead and perform the rendering. Otherwise, a 'page fault' generates a request for new data.

Because the neural network learns by accumulated knowledge over a period of past history, it does not function well until it reaches the point when the history data is enough for gaining this knowledge. We propose a hybrid system, in which heuristic rules are used for prediction in the first few hundred steps and after that, the neural network prediction kicks in. The heuristic rule assumes that the user will continue his/her current moving status, *i.e.,* if moving in one direction, he/she will continue the move in the same direction in the next time step; if the user has been turning, the turning will be continued in the same direction with same speed. A flow chart of this hybrid system is shown in Fig. 6.5.

Given the number of possible movements that can be made during a walkthrough, it is difficult to predict the correct movement every time. In order to speed up overall system performance while tolerating missed predictions, we propose a latency hiding scheme, in which rendering is only performed when the pre-fetching is successful. With this scheme, not all scenes are shown while the user is moving around. However, the system performance is only limited by the speed of the graphics rendering hardware.

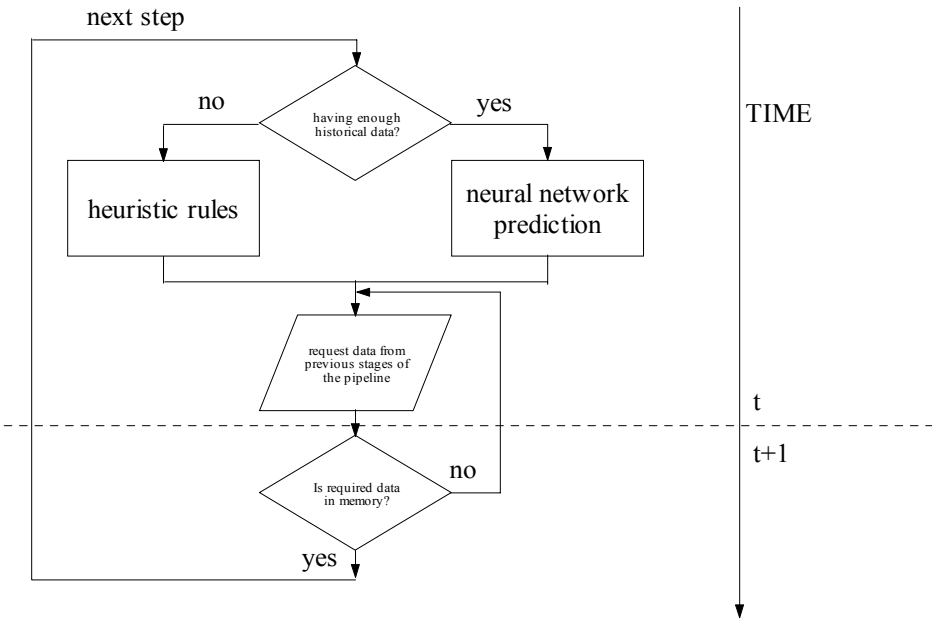


Figure 6.5: The hybrid predictive pre-fetching scheme. User's move at time *t+1* is guessed at time *t*. Before the knowledge base of the neural network is setup, heuristic rules are used in the prediction.

CHAPTER 7

RESULTS AND DISCUSSION

The implementation of the system discussed in this thesis is capable of interactive walkthrough of 1 billion atoms. The time to extract and render one scene is nearly a constant function of the number of atoms. This work was presented at the IEEE Virtual Reality 2002 conference at Orlando, Florida.

Results of performance tests of the techniques described in Chapters 3-6 are discussed in detail in §§7.1-7.4. An overview of the performance in terms of scalability is given in §7.5.

7.1 Data Compression

The compression algorithm has been tested on a Digital Alpha Server 2100 for several configurations. In the case of 46,440 atoms at a temperature of 300K, the original storage scheme required 56 bytes per atom including positions and velocities in double precision and an 8 byte tag used for multiple purpose (atom type, serial number, *etc*). To compress this data, one has to specify the position tolerance, δr, and the time parameter, τ. (The velocity tolerance is computed as $\delta r / \tau$.) Figure 7.1(a) shows the amount of compressed storage per atom as a function of the position tolerance, δr, for three different values of τ. Significant reduction in storage is achieved by increasing δr. However, the information loss due to compression results in an error in the total energy, as is shown in Fig. 7.2(b). Assuming that an energy deviation of 1 K is acceptable, the minimum storage size is

achieved with $\delta r = 5 \times 10^{-3}$ Å and $\tau = 4\,fs$. Figure 7.1(a) demonstrates that the present compression scheme reduces the required storage size by nearly an order-of-magnitude, from 56 to 6.22 bytes/atom. Compared with typical simulation time, the computation time used for compression is found to be negligible.

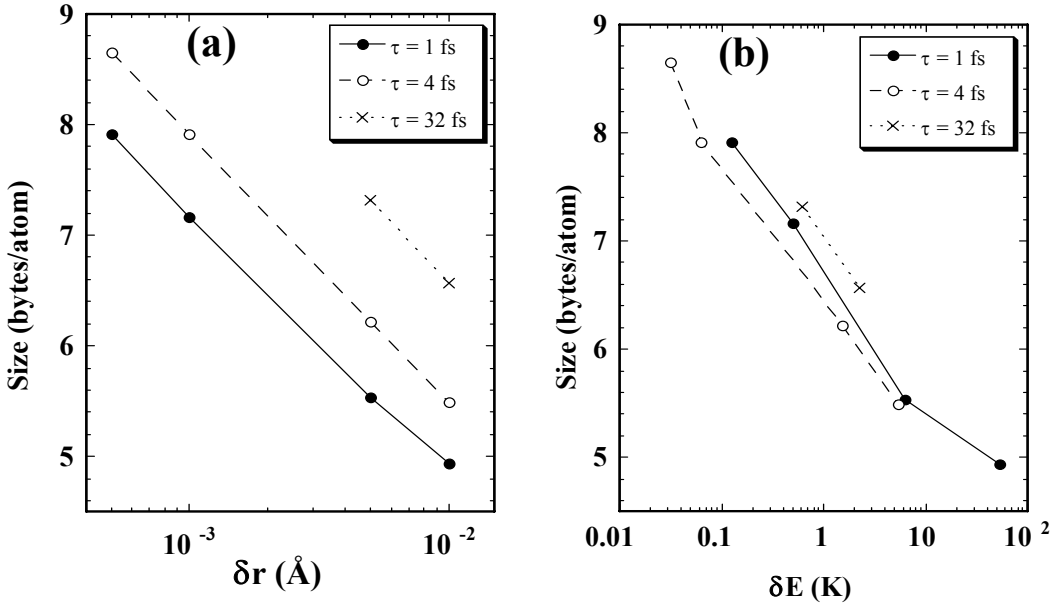


Figure 7.1: Size (per atom) of compressed MD configurations: (a) effect of the tolerance parameters δr and τ, (b) relationship between the storage size and the error in total energy introduced by compression.

7.2 Octree Based View-frustum Culling

Various volume clipping shapes are used to reduce the dataset to be rendered before entering the graphics pipeline of the hardware accelerator card. The final result is a frustum obtained by applying the combination of these clipping schemes as shown in Fig. 7.2.

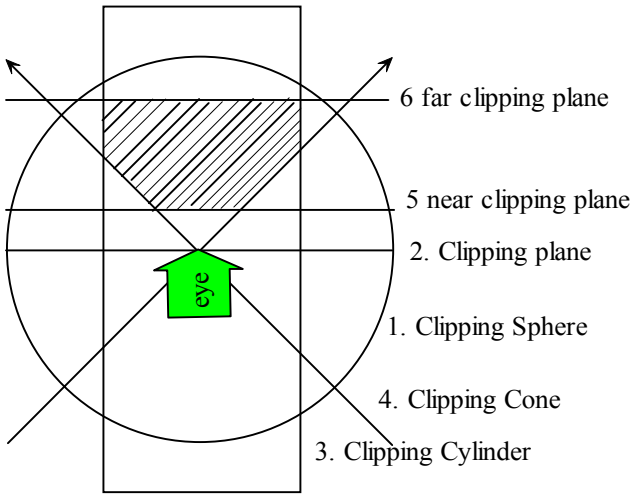


Figure 7.2: Schematic of various clipping operations. The numbers represent clipping sequence. Shaded area is the final frustum.

In a tested configuration involving 1 million atoms, 60% of the atoms are removed after the sphere clipping; about 150,000 to 200,000 atoms will be left after the cylinder clipping, and around 100,000 atoms remain in the final frustum after cone clipping and near/far plane clipping. The shaded area is the final view-frustum. Clipping is performed in the unit of Octree nodes instead of individual atoms in order to speed up the process. This is shown in Table 7.1.

Table 7.1: The effect of clipping in terms of the number of atoms that are removed. Only about 10% of the total atoms remain after these clippings. In a typical configuration of 1 million atoms, 60% are removed after the sphere clipping; about 150,000 to 200,000 will be left after the cylinder clipping and around 100,000 remain in the final frustum after cone clipping and near/far plane clipping.

Sequence of clipping	Percentage of atoms removed
Sphere clipping	60%
Cylinder clipping	20%
Cone clipping	5%
Near/far clipping	5%

Only a fraction of the entire system remains after the dataset reduction phase, as shown in Fig. 7.3; only about 500 octree leaves (about 10% of the total number of octree leaves) remain in the final frustum that will be fed into the graphics rendering pipeline.

Figure 7.4 demonstrates the scalability of the octree algorithm. This figure shows the rendering time per scene as a function of the number of atoms with and without Octree-based visibility culling. It can be seen that as the number of atoms increases, the time taken for octree extraction and atom rendering remain nearly constant.

The efficiency of view frustum culling is related to the position of the viewer. When the viewer is outside the configuration, almost all of the particles are within the view frustum. In this case, the recursive octree culling is turned off, and instead the outer surface nodes of the configuration are output to the next step.

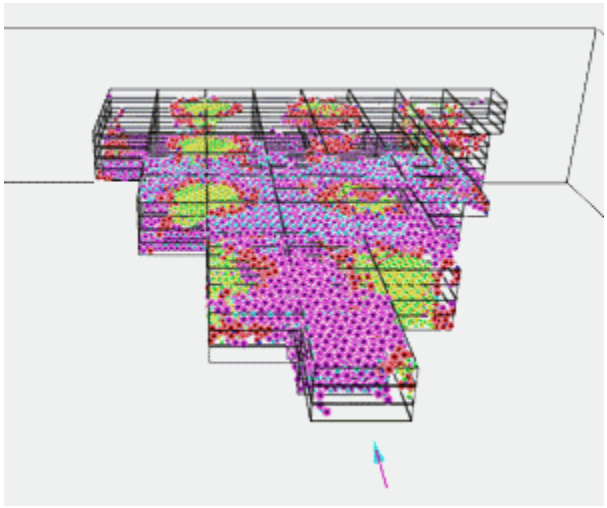


Figure 7.3: Illustration of the Octree dataset reduction. About five hundred Octree leaves remain in the final frustum that goes through the graphics rendering pipeline, with 200 atoms each. The total number of atoms in this case is about 100,000.

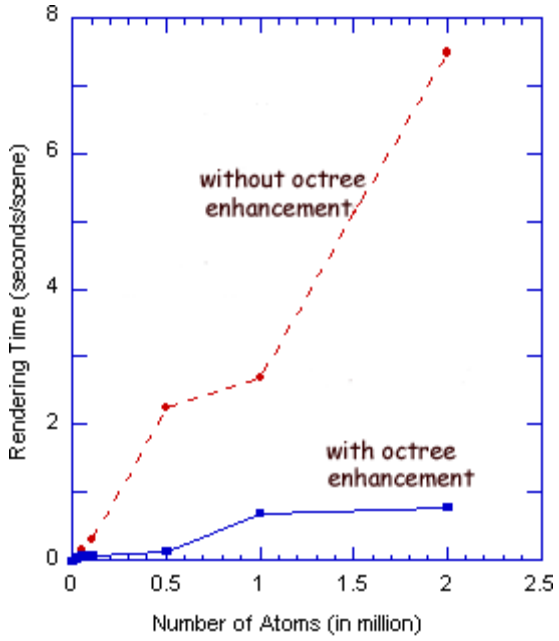


Figure 7.4: Rendering time per scene as a function of the number of atoms with and without Octree based enhancement. The Octree algorithm shows good scalability; when the number of particles increases, the time taken to extract and render the atoms that need to be displayed is almost constant. Lines are guide to eyes.

7.3 Scalable Occlusion Culling

A PC cluster comprised of four 800MHz Intel Pentium III PCs each with 512MB RAM running Red Hat LINUX 6.2 is used for parallel occlusion culling. We have installed SSLeay 0.9.0 [39], OpenLDAP 1.2.7 [40], Globus Toolkit 1.1.3 [41], and MPICH-g2 on both the PCs and an SGI Onyx2 graphics workstation[13], which is directly connected to a immersive rendering device, ImmersaDesk.

The efficiency of occlusion culling depends on radius and density of the particles. When the drawing radius of the particle equals to 1.5Å, 20-25% of the atoms are occluded.

[13] SSLeay is a free implementation of Netscape's Secure Socket Layer, a software encryption protocol. Open Source Lightweight Directory Access Protocol (LDAP) is an open-standard protocol for accessing information services. The protocol runs over Internet transport protocols, such as TCP, and can be used to access stand-alone directory servers. Globus is a research and development project focused on enabling the application of Grid concepts to scientific and engineering computing. Grid is a conceptual framework that conveniently utilizes a collection of computation power as energy power. MPICH-g2 is a Grid-enabled implementation of MPICH developed at the Argonne National Laboratory. MPICH-g2 supports message passing interface (MPI) among heterogeneous platforms.

When the drawing radius increases, more particles are occluded; when the drawing radius is less than 10% of the bond length, less than 1% of occlusion is detected.

Figure 7.5 uses wire frame rendering to show the effect of occlusion culling. The upper and lower pictures are screen captures of the same scene. The upper picture has the occlusion culling feature turned off, while the lower picture shows the effect of occlusion culling. On average, occlusion culling removes about 20-30% of atoms in a scene. In this particular scene, about 65% of atoms were removed in this step, and the frame rate is increased by 3 times.

7.4 Neural Network Pre-fetching

Generalization factor r controls the similarity that an input is considered to be a member of an existing pattern. Different values of r have been tested in several different configurations, and on average the highest hit ratios were achieved for $r = 1.5$.

By employing the CC4 neural network scheme, the system is more responsive to users' input. Figure 7.8 shows that the average prediction hit ratio is 34%, which is much higher than that of an alternative heuristic method, 24%. This indicates that one third of the time, the user feels significantly improved system response. Combined with the proposed latency hiding scheme, the system will only render on average about 1 frame per three frames, but the user will experience apparent performance gain in terms of interactivity.

The radial basis network [42] and backpropagation network [36] are tested for purposes of performance comparison. Table 7.2 shows that neither of them is suitable for the interactive application in this work because of the excessive training time required. This test is conducted with MATLAB on an 800MHz Pentium III PC running Windows 2000 Professional. Time shown for radial basis and backpropagation neural networks are the

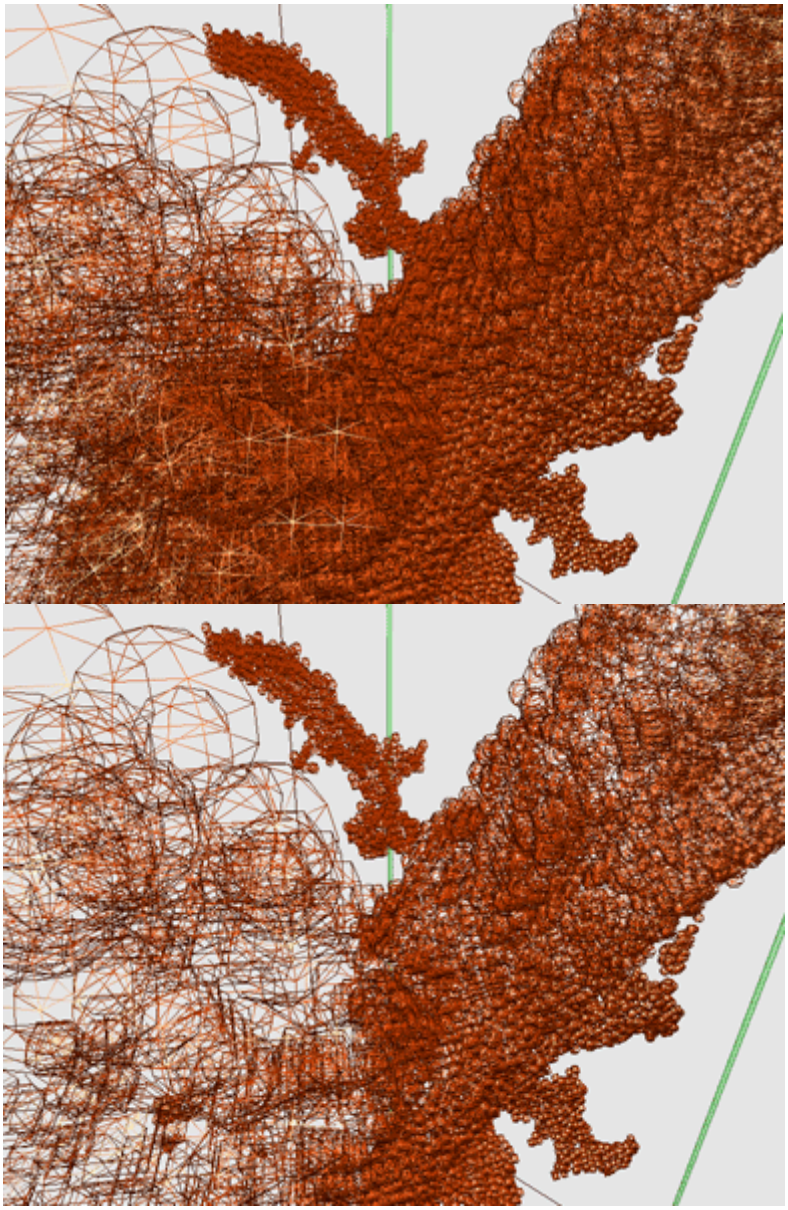


Figure 7.5: Wire frame pictures showing effect of occlusion culling. These are two screen captures of the same configuration. Occlusion culling is turned off in the upper scene. The lower scene shows the effect of occlusion culling. On average, occlusion culling removes about 20-30% of atoms in a scene. In this particular case, about 65% of atoms were removed in this step, and the frame rate is increased by 3 times. There is no visible difference in the surface rendering scene.

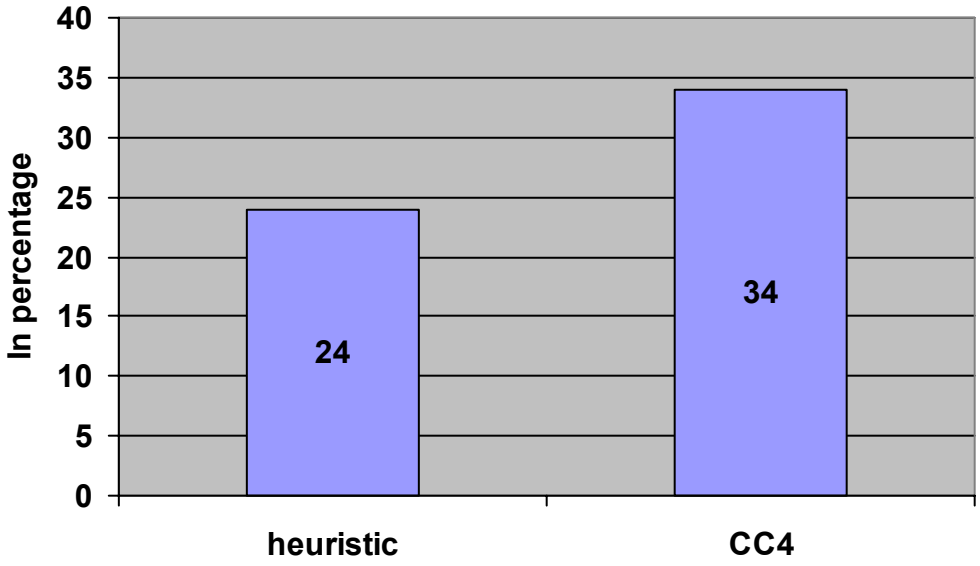


Figure 7.6: Hit ratio (in percentage) comparison of CC4 algorithm and heuristic method. The CC4 achieved 34% hit ratio, which means in one third of the time, the system can guess the users' next move correctly.

training time, while the time shown for the CC4 neural network is the total of training time and generalization time. Similar to the generalization factor *r*, the spread parameter of radial basis function neural network indicated the selectivity of the neuron. Spreads tested for radial basis neural network are 0.8, 1, 1.2, 2; the default value 1.0 turns out to be optimal. The training goal for backpropagation is 10% SSE or 10,000 epochs, or degrading < $1e^{-14}$. It is clear that neither of the neural networks is suitable for interactive applications. We have found no literature that shows any other types of neural network capable of instantaneous training.

Table 7.2: Performance comparison of three neural networks. The time shown for radial basis neural network and back propagation neural network are the training time. The time shown for the CC4 neural network is the total of training and generalization time. Radial basis and backpropagation, as well as other type of neural networks which are not listed here, are not suitable for interactive applications.

	Radial Basis	CC4	Backpropagation
Hit-ratio (%)	20	34	36
Time (second)	20	0.3	180

7.5 Scalability Test

We have performed the scalability test on our visualization system.

To study the effect of the Octree-based occlusion culling, we first compare the rendering times as a function of the number of particles with and without Octree enhancement with serial implementation. First, a serial version without any of the techniques described in this thesis was run on a standalone SGI Onyx2. The blue line in Fig. 7.7 shows the rendering time of this sequence. It takes more than 1 second to render one frame when the number of atoms is 20,000. When the number of atoms is 100,000, the time to render one

frame is about 3 seconds. Above this size, the time to render one frame increases rapidly. It takes about 8 seconds to render one frame for 120,000 atoms, which is too long for any interactive application. The second sequence of runs (shown in green) incorporated the Octree data management enhancement on a standalone SGI Onyx2. Up to 100,000 atoms are rendered at less than 1 second per frame. Rendering of 1 million atoms takes about 5 seconds per frame. To study the scalability of the parallel/distributed implementation, we have performed the third sequence of performance tests on the SGI Onyx2 graphics workstation and four 800 MHz Intel Pentium III PCs running Linux Redhat 6.2. The red line in Figure 7.7 also shows the rendering time of this sequence as a function of the number of particles. It takes 0.7 second to render one frame of 100,000 atoms. The time to render 1 frame increases only slightly when the size of the dataset increases from 100,000 to 1 billion. Time to render one frame for 1 billion atoms is about 1.2 seconds. Together with latency hiding scheme, the response time can be improved by up to 3 times at the cost of dropping frames.

Figure 7.7 clearly demonstrates that the parallel visualization scheme described in this dissertation is scalable for interactive walkthrough of particle systems of size up to 1 billion.

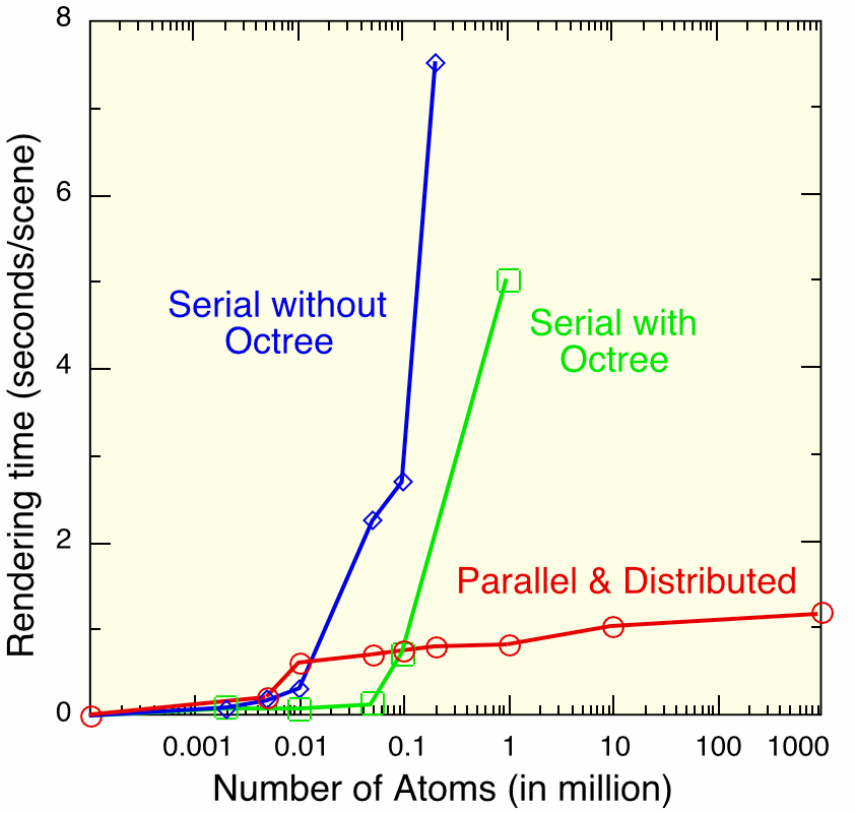


Figure 7.7: This figure summarizes the overall result of this research by plotting rendering time per frame as a function of the number of particles showing the improvement over previous attempts. For 1 billion atoms, about 1 second per frame is rendered. Three lines with different colors show scalability of the system of different versions of the implementation with speed up techniques added in time. Marked points are running instances. Serial version (blue line) with Octree data management enhancement blows out when atoms number is greater than 10 thousands. After adding the Octree data management system (green line), 100 thousands atoms can be handled in reasonable responding time (less than 1 second per frame). The final version (red line) of this work can handle 1 billion atoms at about 1 second per frame. Together with latency hiding scheme, the responding time can be improved by up to 3 times at the cost of dropping frames.

CHAPTER 8

CONCLUSIONS

This thesis has described an approach to rendering large numbers of particles interactively. Both a conceptual framework and implementation details are given for our comprehensive solutions towards billion particles walkthrough. The scalability of the system has been demonstrated for up to 1 billion particles.

We have demonstrated thus in this work that billion particle interactive walkthrough is a workable problem given current computing resources.

We have achieved the goal by splitting the work between a traditional high-end graphics workstation and a Linux PC cluster, and by dividing the data among nodes of the PC cluster. We have also developed a parallel occlusion culling scheme. We have suggested a neural network based behavior prediction mechanism in the field of scientific visualization for the first time.

Specific techniques developed in this work include the following:

1. An error bound controlled compression scheme has been developed, based on a space filling curve. The method of Z-curve compression uses little system resources to compute, while preserving the precision of molecular dynamics simulation in terms of fluctuation of the total energy.
2. An Octree tree data management system has been employed to efficiently perform data managing and view frustum culling.
3. A parallel occlusion culling scheme has been developed to efficiently perform occlusion culling. The difference of this method compared with other existing schemes is that the

computing is based on the resolution of display equipment, without working on unnecessary precisions that have no impact on final display. Computational complexity is thus related to limited resolution.

4. A neural network based behavior prediction has been adopted. By adopting a special training algorithm, the excessive training time associated with normal neural network implementations has been eliminated. To our knowledge, this is the first application of a machine learning method to scientific visualization of particle data. Together with the latency hiding scheme we have developed, the response time is greatly reduced, and the user can feel a significant speed up in navigating the immersive environment.

The visualization system developed in this dissertation works for interactive walkthrough of a billion particle dataset. I expect that the system can be further enhanced and made applicable to broader areas through the following developments.

1. A quantified study of system bottleneck. For an interactive application, the most important measurement is the time to draw a single frame, which can be divided into partial times to perform occlusion culling, view frustum culling, physical rendering, and communication costs as shown in Fig. 6.1. If the required time need for each component is known, we can not only direct our optimization efforts more efficiently, but also predict the scalability of the current system before undergoing major architecture changes. It is also important to study the relationship between the number of nodes on the PC cluster and the speed up that can be achieved. It would be beneficial to derive a formula for evaluating parallel rendering schemes.
2. Test of different parallel schemes. I have several different ideas to implement parallel rendering in an interactive walkthrough environment, and would like to test their

performance. These ideas are: i) Tasks are divided among a number of nodes, where each node locally stores particles' properties and also has information about its close neighbors; ii) each node acts as an intelligent agent which handles visibility culling within its boundary, as well as forming dynamic groups with neighboring nodes; iii) the result of one of these groups is streamed to the display equipment.

3. Develop a PC-based front end client node. The ImmersaDesk virtual reality equipment this work has been based on is rather expensive and many researchers cannot take advantage of it. A PC-based front end with stereo goggles will be a useful alternative for a broader class of researchers. A wider variety of interactivity in terms of more input devices compatibility is desired.
4. Use a Linux box PC to replace the SGI Onyx graphics system. One reason is that Linux is inexpensive to acquire and easy to maintain; another reason is that SGI may not be a reliable long-term supplier of high end graphics workstations under the current economic conditions.
5. Study the load balancing problem. Load balancing was not given much consideration in this work because the configuration we were interested in, such as the fiber composite dataset, are nearly unified in terms of density. When we extend it to more application areas, such as biomedical and micro-mechanical systems, local balancing will have more impact on the overall system performance, and therefore require more attention.
6. Extension to other areas such as bioinformatics. Problems such as protein folding and cell biology simulations are promising application areas of this work. In some cases, this work can be applied with little adjustment, such as the 3D reconstruction of optical

nerve heads under pressure, which is underway in the Biological Computation and Visualization Center at LSU; in other cases, more modifications are needed.

REFERENCES

1. Hockney, R.W. and J.W. Eastwood, Computer Simulations Using Particles. 1988, Bristol: Adam Hilger Ltd.

2. McCormick, B.H., T.A. DeFanti, and M.D. Brown, Visualization in Scientific Computing. Computer Graphics, 1987. **21**: p. 1-14.

3. Smith, P.H. and J.V. Rosendale, eds. Data Visualization Corridors: Report on the 1998 DVC Workshop Series. 1998.

4. Vashishta, P., R.K. Kalia, and A. Nakano, Large-Scale Atomistic Simulations of Dynamic Fracture. Computing in Science & Engineering, 1999. **1**(5): p. 56-65.

5. Fakespace Systems Inc. http://www.fakespace.com.

6. Shimojo, F., et al., A Scalable Molecular-Dynamics Algorithm Suite for Materials Simulations: Design-Space Diagram on 1024 Cray T3E processors. Future Generation Computer Systems, 2000. **17**: p. 279-291.

7. Manssour, I.H., et al., Visualizing and Exploring Meteorological Data using a Tool-Oriented Approach, in Visualization & Modeling. 1997, Academic Press: London. p. 16.

8. Taylor, V.E., et al., Visualization of Finite Element Simulations: Identifying and Reducing the Critical Lag Components. IEEE Computer Graphics and Application, 1996: p. 42-43.

9. Zheng, J., Parallel Visualization, X. Liu, Editor. 2001.

10. Humphreys, G. and P. Hanrahan. A Distributed Graphics System for Large Tiled Display. in IEEE Visualization '99. 1999. San Fransisco, CA.

11. Buck, I., G. Humphreys, and P. Hanrahan. Tracking Graphics State For Networked Rendering. in SIGGRAPH Eurographics Graphics Hardware Workshop. 2000. Switzerland.

12. Ma, K.-L. and D.M. Camp. High Performance Visualization of Time-Varying Volume Data over a Wide-Area Network. in High Performance Networking and Computing Conference. 2000. Dallas, TX.

13. Shen, H., L. Chiang, and K. Ma. A Fast Volume Rendering Algorithm for Time-Varying Fields Using a Time-space Partition(tsp) Tree. in IEEE Visualization Conference. 1999.

14. Bethel, W., et al. Using High-Speed WANs and Network Caches to Enable Distributed Visualization. in High Performance Networking and Computing Conference. 2000. Dallas, TX.

15. Bethel, W., Visualization Dot Com. IEEE Computer Graphics and Application, 2000.

16. Bethel, W., et al. Visapult - A Prototype Remote and Distributed Visualization Application and Framework. in SIGGRAPH. 2000. New Orleans, LA.

17. Aliaga, D., et al., A Framework for the Real-Time Walkthrough of Massive Models. 1998, University of North Carolina: Chapel Hill.

18. Zhang, H., Effective Occlusion Culling for the Interactive Display of Arbitrary Models, in Department of Computer Science. 1998, University of North Carolina: Chapel Hill.

19. Erikson, C.M., Hierarchical Levels of Detail to Accelerate the Rendering of Large Static and Dynamic Polygonal Environments, in Department of Computer Science. 2000, University of North Carolina: Chapel Hill. p. 183.

20. Cohen-Or, D., Y. Chrysanthou, and C. Silva. A Survey of Visibility for Walkthrough Applications. in ACM SIGGRAPH 2000. 2000. New Orleans, LA.

21. Omeltchenko, A., A data-compression algorithm for large-scale molecular-dynamics simulations on parallel computers. 1997, Louisiana State University: Baton Rouge.

22. Saona-Vazquez, C., I. Navazo, and P. Brunet, The Visibility Octree: A Data Structure for 3D Navigation. Computer & Graphics, 1999. **23**(5): p. 635-644.

23. Omeltchenko, A., et al., Scalable I/O of large-scale molecular dynamics simulations: A data-compression algorithm. Computer Physics Communications, 2000. **131**: p. 78-85.

24. Salomon, D., Data Compression. 1997, New York: Spinger-Verlag.

25. Thompson, T., An Inside Look at the Most Popular 3-D Environment: OpenGL, QuickDraw 3D, and Direct3D, in BYTE. 1996.

26. Pabst, T. and S. Orozco, ATi's New Radeon - Smart Technology Meets Brute Force. 2000.

27. Charlton, E.F., An Octree Solution to Conservation-laws over Arbitrary Regions (OSCAR) with Applications to Aircraft Aerodynamics, in Aerospace Engineering and Scientific Computing. 1997, University of Michigan.

28. Kelleghan, M., Octree Partitioning Techniques. Game Development, 1997.

29. Sharma, A., et al., Million Atom Walkthrough: Octree-based Fast Visibility Culling and Multiresolution Rendering for Scalable Atomistic Visualization. 2001.

30. Shoaff, W., The Hidden/Visible Object Problem.

31. Bajaj, C. and S. Cutchin. Web Based Collaborative Visualization of Distributed and Parallel Simulation. in IEEE Parallel Visualization and Graphics Symposium. 1999. San Fransciso.

32. Sharma, A., X. Liu, and e. al. Immersive and Interactive Exploration of Billion-Atom System. in IEEE Virtual Reality 2002 Conference. 2002. Orlando, FL.

33. Brown, R.G., Smoothing, Forecasting and Prediction of Discrete Time Series, Management and Quantitative Methods Series. 1963, Englewood Cliffs, NJ: Prentice-Hall.

34. Thrun, S. and L. Langford, Monte Carlo Hidden Markov Models. 1998, Carnegie Mellon University: Pittsburgh, PA.

35. Mozer, M.C., Neural Net Architectures for Temporal Sequence Processing. Predicting the Future and Understanding the Past, ed. A. Weigend and N. Gershenfeld. 1994, Redwood City, CA: Addison-Wesley.

36. Fu, L., Neural Networks in Computer Intelligence. 1994, New York: McGraw-Hill.

37. Tang, K.-W. and S.C. Kak, A new corner classification approach to neural network training. Circuits Systems Signal Processing, 1998. **17**(4): p. 459-469.

38. Shu, B. and S.C. Kak, A Neural Network Based Intelligent Metasearch Engine. Information Sciences, 1999. **120**: p. 1-11.

39. Hirsch, F.J., Introducing SSL and Certificates using SSLeay. World Wide Web Journal, 1997. **2**(3).

40. Open Source for LDAP software and information. http://www.openldap.org.

41. The Globus Project. http://www.globus.org.

42. Blanzieri, E., Learning Algorithm for Radial Basis Function Networks: Synthesis, Experiments and Cognitive Modelling, in Department of Computer Science. 1998, University and Polytecnic Turin: Turin.

VITA

Xinlian Liu was born in Tianjin, China, on October 24th, 1971, the son of Kezhong Liu and Huifang Yang. After completing his work at Nankai High School, he went on to Huazhong University of Science and Technology, where he studied computer peripherals and received his Bachelor of Engineering degree in May 1993. During the following years, he was employed as an assistant engineer with the Data Processing Division of the National Meteorological Center in Beijing, China. In August 1996, he entered the Graduate School of Louisiana State University and Agricultural & Mechanical College, Baton Rouge, Louisiana. Currently he is a candidate for the degree of Doctor of Philosophy in computer science, which will be conferred at the August 2002 commencement ceremony. Upon graduation, he will be working as a postdoctoral fellow at the Argonne National Laboratory in Chicago, Illinois.